

APPENDIX E

```

#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include "fastdb.h"

int InsertRecordIndex ( int keyfd, char *key, unsigned long offset );

keyindx_t  newbucket[CRCBUCKETSIZE+1];

//
// Name:      mkindx
//
// Description:
//      Program to create double hash CRC index files from
//      a Data file. This index file is used by getrec
//      for testing QueryRecord() performance.
//
//      Creates "keyindx", a double hash CRC index file.
//
main( int argc, char *argv[] )
{
    char    buf[4096];
    int bytes;
    int lines;
    FILE    *fdata;
    char    *ptr;
    int keyfd;
    int reclen;
    int i;
    unsigned long    offset;
    keyindx_t    key;

    if ( argc != 2 || *argv[1] == '-' )
    {
        printf ( "Usage: %s <filename>\n", argv[0] );
        exit(1);
    }

    // initialize CRC bucket
    for ( i = 0; i < CRCBUCKETSIZE + 1; i++ )
    {
        newbucket[i].crc16 = 0;
        newbucket[i].offset = MAXOFFSET;
    }

    // open Data file to build index from...
    fdata = fopen ( argv[1], "r" );
    if ( !fdata )
    {
        printf ( "Can't open %s for read. %s\n",
            argv[1], strerror ( errno ) );
        exit(1);
    }

```

```

    }

    // preallocate 2**16 keyindx file buckets in key index file
    keyfd = open ( "keyindx", O_RDWR | O_CREAT | O_TRUNC, 0640 );
    if ( !keyfd )
    {
        printf ( "Can't open keyindx for write. %s\n",
                strerror ( errno ) );
        exit(1);
    }

```

```

    // initialize CRC index file
    for ( i = 0; i < CRCTABLESIZE; i++ )
    {
        if ( write ( keyfd, (char *)newbucket, sizeof(newbucket) )
            != sizeof(newbucket))
        {
            printf ( "write to keyindx file failed\n" );
            exit(1);
        }
    }

```

```

    lines = 0;
    offset = 0;
    for ( offset = 0; ; offset += (unsigned long) reclen )
    {
        // get next record
        ptr = fgets ( buf, sizeof(buf), fdata );
        if ( !ptr )
            break;

        // save this record size for offset update
        reclen = strlen ( buf );

        // zap newline
        ptr = strchr ( buf, '\n' );
        if ( ptr )
            *ptr = 0;

        // get email address, just happens to be first field.
        ptr = strchr ( buf, '\t' );
        if ( ptr )
            *ptr = 0;

        // Add record to index file...
        InsertRecordIndex ( keyfd, buf, offset );
        lines ++;
    }

```

```

    close ( keyfd );
}

```

```

//
// InsertRecordIndex adds an index entry for this record's offset.
//
int InsertRecordIndex ( int keyfd, char *key, unsigned long offset )
{

```

```
int i;
unsigned long startoff;
unsigned long curpos;
unsigned short crc;
unsigned short crc16;
keyindx_t  crcbucket[CRCBUCKETSIZE+1];
keyindx_t  *keyptr;

// calculate CRC-CCITT checksum
crc = 0;
crcstr ( &crc, (unsigned char *) key );

// calculate CRC-16 checksum
crc16 = 0;
crc16str ( &crc16, (unsigned char *) key );

// find right bucket
startoff = (unsigned long)crc * (CRCBUCKETSIZE + 1) * sizeof (keyindx_t);

while ( 1 )
{
    // seek to this bucket's offset
    curpos = lseek ( keyfd, startoff, SEEK_SET );
    if ( curpos != startoff )
    {
        printf ( "lseek in keyindx file failed\n" );
        exit(1);
    }

    // read the bucket
    if ( read ( keyfd, (char *)crcbucket, sizeof(crcbucket) )
        != sizeof(crcbucket) )
    {
        printf ( "read from keyindx file failed\n" );
        exit(1);
    }

    // search for empty slot
    keyptr = crcbucket;
    for ( i = 0; i < CRCBUCKETSIZE ; i++, keyptr++ )
    {
        // check for empty slot in bucket
        if ( keyptr->offset == MAXOFFSET && keyptr->crc16 == 0 )
        {
            // found empty slot, use it
            keyptr->offset = offset;
            keyptr->crc16 = crc16;
            lseek ( keyfd, startoff + (i * sizeof (keyindx_t)), SEEK_SET );
            if ( write ( keyfd, (char *)keyptr, sizeof (keyindx_t) )
                != sizeof (keyindx_t) )
            {
                printf ( "write entry to keyindx file failed\n" );
                exit(1);
            }
        }
    }
    return ( 1 );
}
```

```

        // check for possible duplicate record
        if ( keyptr->crc16 == crc16 )
            printf ("possible duplicate crc = 0x%X crc16 = 0x%X key = %s\n",
                    crc, crc16, key );
    }

    // no empty slot found, check if next crcbucket is linked.
    if ( keyptr->offset == MAXOFFSET )
        break; // no more buckets

    // follow link to next bucket...
    startoff = keyptr->offset;
    //printf ( "Linking to next bucket at 0x%x\n", startoff );
    } // while ( 1 )

    // ran out of space, need to allocated and link new bucket
    //printf ( "Allocating new bucket for crc = 0x%x\n", crc );
    keyptr->offset = lseek ( keyfd, 0, SEEK_END );

    // initialize new CRC bucket
    newbucket[0].offset = offset;
    newbucket[0].crc16 = crc16;

    // write new bucket to end of key index file
    if ( write ( keyfd, (char *)newbucket, sizeof(newbucket) )
        != sizeof(newbucket))
    {
        printf ( "write bucket to end of keyindx file failed\n" );
        exit(1);
    }

    // update previous bucket's next pointer to this new bucket's offset
    lseek ( keyfd, startoff + (i * sizeof (keyindx_t)), SEEK_SET );
    if ( write ( keyfd, (char *)keyptr, sizeof (keyindx_t) )
        != sizeof (keyindx_t))
    {
        printf ( "write entry to keyindx file failed. %s\n",
                strerror ( errno ) );
        exit(1);
    }

    return ( 0 );
}

```